1998

VLSI Architecture: Past, Present, and Future

William J. Dally and Steve Lacy
Computer Systems Laboratory
Stanford University
Stanford, CA 94305
{billd,slacy}@csl.stanford.edu

Abstract

This paper examines the impact of VLSI technology on the evolution of computer architecture and projects the future of this evolution. We see that over the past 20 years, the increased density of VLSI chips was applied to close the gap between microprocessors and high-end CPUs. Today this gap is fully closed and adding devices to uniprocessors is well beyond the point of diminishing returns. To continue to convert the increasing density of VLSI to computer performance we see little alternative to building multicomputers. We sketch the architecture of a VLSI multicomputer constructed from c. 2009 processor-DRAM chips and outline some of the challenges involved in building such a system. We suggest that the software transition from sequential processors to such fine-grain multicomputers can be eased by using the multicomputer as the memory system of a conventional computer.

1. Introduction

Twenty years ago at the first conference in this series there was consensus that the best way to apply VLSI technology to information processing problems was to build parallel computers from simple VLSI building blocks. Four of the seven papers in the architecture session addressed this topic [Sequin79, Mago79, Brown79, Davis79] including two papers on tree machines, one on cellular automata, and one on dataflow. Considerable research over the past two decades focused on the design of parallel machines and many valuable research contributions were made. The mainstream computer market, however, was largely unaffected by this research. Most computers today are uniprocessors and even large servers have only modest numbers (a few 10s) of processors.

The situation today is different. We again anticipate an exponential increase in the number of devices per chip. However, unlike 1979, there are few opportunities remaining to apply this increased density to improve the performance of uniprocessors. Adding devices to modern processors to improve their performance is already well beyond the point of diminishing returns. There are few credible alternatives to using the increased device count other than to build additional processors. Because the chip area of contemporary machines is dominated by memory, adding processors (without adding memory) boosts efficiency by giving a large return in performance for a modest increase in total chip area. To realize the potential of such fine-grain machines to convert VLSI density into application performance we must address challenges of locality, overhead, and software.

The remainder of this paper gives the authors'perspective on VLSI architecture. We begin in Section 2 by reviewing the situation at the first VLSI conference in 1979. We show that the parallel revolution predicted then was not immediately forthcoming because the increased density of VLSI was applied instead to closing the gap between microprocessors and high-end CPUs. In Section 3 we compare the situation today to that of 1979. We see that today there is no gap:

high-end CPUs are microprocessors, and there are few alternatives to explicit parallelism. We sketch our vision of a VLSI architecture in 2009 in Section 4. The system is built from 4Gb DRAM chips each of which contains 64 small but powerful processors. The fundamental challenges involved in building such a system are discussed in Section 5 and we touch on the problem of parallel software in Section 6.

2. Twenty years of VLSI architecture

In 1979[1] anticipated scaling of VLSI technology favored the development of *regular* machines that exploited *concurrency* and *locality* and that were *programmable*. As shown in columns 2 and 3 of Table 1, twenty years was expected to bring more than a thousandfold increase in the number of grids[2], and hence the number of devices that could be economically fabricated on a chip. Clearly concurrency (parallelism) would need to be exploited to convert this increase in device count to performance. Locality was required because the wire bandwidth at the periphery of a module was scaling only as the square root of the device count, much slower than the 2/3 power required by Rent's rule [LanRus71]. Also, even in 1979 it was apparent that wires, not gates, limited the area, performance, and power of many modules. The issue of design complexity motivated regularity and programmability. Designing an array of identical, simple processing nodes is an easier task than designing a complex multi-million transistor processor. A programmable design was called for so that the mounting design costs could be amortized over large numbers of applications.

Table 1: Semiconductor Technology in 1979, 1999, and 2019

Parameter	1979	1999	2019	Units
Gate length	5	0.2	0.008	μm
Gate delay	3000	150	7.5	ps
Clock cycle	200	2.5	0.08	ns
Gates/Clock	67	17	10	
Wire pitch	15	1	0.07	μm
Chip edge	6	15	38	mm
Grids/Chip	1.6×10^5	2.3×10^8	3.0×10^{11}	

In the twenty years since the first conference many of the hard problems of parallel machine design have been solved. We now understand how to design fast, efficient networks to connect arrays of processors together [Dally92, DYN97][3]. Mechanisms that allow processors to quickly communicate and synchronize over these networks have been developed [LDK+98]. We understand how to implement efficient, coherent shared memory systems [ASHH88]. Several meth-

1. Data similar to those in Table 1 were presented in Mead's paper at the first conference [Mead79].
2. A grid is an intersection of a horizontal and vertical wire. Hence the number of grids on a chip is the square of the number of wiring tracks that fit along one edge of a chip. As VLSI chips are limited by wiring, not devices, the number of grids is a better measure of complexity than the number of transistors.
3. The key here is to match the design of the network to the properties of the implementation technology rather than to optimize abstract mathematical properties of the network.

ods of programming parallel machines have been demonstrated. Research machines were constructed to demonstrate the technology, provide a platform for parallel software research, and solve the engineering problems associated with its realization [Seitz85, NWD93, SBSS93]. The results of this research resulted in numerous commercial machines [Scott96] that form the core of the high-end computer industry today[4].

Just as important, we learned what didn't work: MIMD machines are preferable to SIMD machines even for data-parallel applications. Similarly, general-purpose MIMD machines are preferable to systolic arrays, even for regular computations with local communication. Bit-serial processors loose more in efficiency than they gain in density[5]. A good general-purpose network (like a 3-D torus) usually outperforms a network with a topology matched to the problem of interest (like a tree for divide and conquer problems). It is better to provide a general-purpose set of mechanisms than to specialize a machine for a single model of computation.

While successful at the high end, parallel VLSI architectures have had little impact on the mainstream computer industry. Most desktop machines are uniprocessors and even departmental servers contain at most a few 10s of processors. Today's mainstream microprocessor chips are dense enough to hold 1000 of the 8086s or 68000s of 1979, yet we use all of this area to implement a single processor.

What went wrong? Why isn't the average PC a fine-grain parallel machine with 10s of processors on integrated processor-DRAM chips in the spirit of Mosaic or the J-Machine? By many objective measures this would clearly be a more *efficient* architecture.

There are three main reasons for this course of events:

1. There was considerable opportunity to apply additional grids to improve the performance of sequential processors.
2. Software compatibility favored sequential machines.
3. High-overhead mechanisms used in early parallel machines motivated a coarse granularity of both hardware and software.

In 1979 there was more than a factor of 100 difference in performance between the best microprocessors (0.5MIPS, 0.001MFLOPS) and a high-end CPU such as used in the Cray 1 (70MIPS, 250MFLOPS [Russel78]) or IBM 370. Only a small part of this difference, about a factor of 3, was due to the difference in gate delay between bipolar and MOS technology. Most of the difference was due to increased gate count that was used to aggressively pipeline execution and to exploit parallelism.

Between 1979 and 1999 microprocessors closed this gap by incorporating most of the advanced features pioneered in mainframes and supercomputers in the 1960s and 70s as well as a few new tricks. On-chip caches, on-chip memory management units, pipelined multipliers and floating-point units, multiple instruction issue, and even out-of-order instruction issue were added to processors during this period. The addition of these features, along with quadrupling the word width from 16-bits to 64-bits created a sufficient appetite for grids without resorting to explicit parallelism.

During the past 20 years, the performance of a high-end microprocessor increased from 0.5MIPS to 500MIPS, about a factor of 1000. From the data in Table 1, we see that clock frequency increased by a factor of 80: a factor of 20 is due to gate delay and a factor of 4 is due to reducing the number of gates per clock. The remaining factor of 12.5 reflects a reduction in clocks per instruction (CPI) from about 10 for the unpipelined microprocessors of 1979 to just under 1 for today's 3- and 4-way multiple-issue superscalar processors.

---

4. Many important contributions were made beyond the few that I cite here.
5. They are an example of processors that are not yet to the point of diminishing returns.

Software evolved considerably during the last 20 years: from text-based applications running on proprietary operating systems (like VMS and MVS) to graphics-based applications running on third-party operating systems (like Windows and Unix). What remained constant, however, was the sequential nature of this software. Manufacturers wanting to sell machines that would run existing software needed to build fast sequential machines.

Commercial parallel machines shut themselves out of the mainstream by taking a path that emphasized capability (running very large problems) rather than economy (solving the most problems per dollar × second). These machines were coarse-grained both in the amount of memory per node and in the size of individually scheduled tasks. Early machines were forced by high-overhead mechanisms to run programs with large tasks sizes. To ensure software compatibility, later machines were forced to follow this same route, often because of macro packages (like PVM and MPI) that hid the improved mechanisms behind high-overhead software. A coarse-grain parallel computer node is largely indistinguishable from a conventional workstation or PC with one exception: it is considerably more expensive. While one can equalize the expense by constructing coarse-grain parallel computers from networks of workstations [ACP95], achieving an economy that is better than serial machines requires fine-grain nodes [FKD+95].

In summary, for most of the 80s and 90s software compatibility motivated building sequential machines; there was little economic advantage to coarse-grain parallel machines; and there were many obvious ways to use more grids to make a sequential CPU faster. Given this environment, it is no surprise that industry responded by making sequential CPUs faster and only building coarse-grain parallel machines.

3. The next 20 years

The next 20 years promise to be exciting ones in the area of VLSI architecture with a major revolution in the architecture of mainstream processors. Continued scaling of technology (columns 3 and 4 of Table 1[6]) will give us yet another thousandfold increase in chip density. As in 1979 it is natural to think of developing architectures that are *programmable* and exploit *concurrency* and *locality* to exploit this increased density. Unlike 1979, however, there are three reasons why a revolution is likely now:

First, sequential processors are out of steam. While clever architects will undoubtedly continue to develop new methods to squeak a few percentage points more performance from sequential processors, we are clearly well past the point of diminishing returns[7]. Large amounts of chip area are spent on complex instruction issue logic and branch prediction hardware while yielding small improvements in performance[8]. To continue improving performance geometrically each year, there is no alternative except to exploit explicit parallelism.

---

6. The 1997 SIA roadmap only extends to 2010. For symmetry, I have optimistically extrapolated these numbers to 2019.
7. To be precise, the point of diminishing returns is where the incremental return in performance per unit cost (area), $\frac{\partial P}{\partial A}$, drops below the incremental return for an alternative expenditure of area. Assuming linear speedup, for example, this point happens when $\frac{\partial P}{\partial A} < \frac{P}{A}$. For actual systems, with sublinear speedup, it happens slightly later [Dally86].
8. Ironically, these heroic efforts to extend instruction-level parallelism (ILP) usually involve converting data parallelism, which is easy to exploit on a multicomputer, to ILP which is hard to exploit on a uniprocessor, for example by loop unrolling.

Second, technology scaling is rapidly making wires, not transistors, the performance limiting factor. Each time line widths halve, the devices get approximately twice as fast and a minimum width wire of constant length gets four times slower. Contemporary architectures, that depend on global control, global register files, and a linear memory hierarchy, don't scale well with large wire delays. Instead, these slow wires motivate architectures that exploit locality[9] by operating on data near where it is stored. Architectures that distribute a number of simple yet powerful processors throughout the memory, for example, are capable of exploiting this type of locality.

Finally, the cost[10] of machines today is dominated by memory. Twenty years of increasing the memory capacity of machines to match the performance of the microprocessor has left us with computers that are (in terms of silicon area) mostly memory. A competent 500MHz, pipelined, in-order, dual-issue processor with floating point can be built today in an area of about 10M grids (about 1/20 of a modern 0.2μm chip). In contrast, a memory system with a 512MByte main memory and a 512KByte cache takes about 20 200M grid chips. Such a machine has a 400:1 ratio of memory to processor area.

A parallel machine can *turn over* its expensive memory system more quickly than a sequential machine with the same amount of memory and hence can offer more economical operation. That is, the parallel machine ties up less time on the costly memory per problem and hence solves more problems per dollar× second. By adding inexpensive, yet capable, processors without adding more memory, the cost of the machine is increased modestly while its performance is multiplied considerably, even if speedup is far less than linear.

4. Sketch of a Fine-Grain Machine

Figure 1: Sketch of a fine-grain machine using 2009 technology.

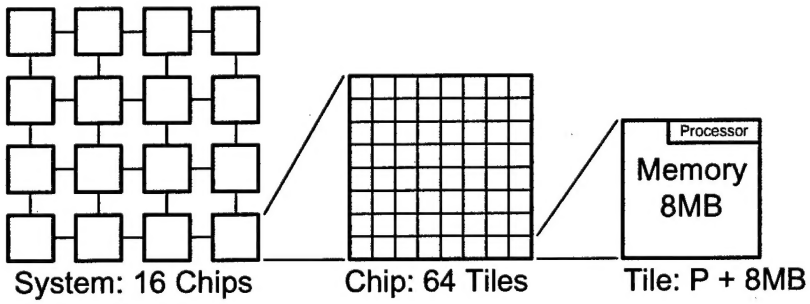

Figure 1 and Table 2 illustrate the type of fine-grain machine that might be constructed using 0.1μm CMOS technology in about ten years' time. The system consists of a number of integrated processor-memory chips connected by a direct interconnection network using high-speed links. Each chip is divided into 64 *tiles* each of which contains a powerful processor (64-bit dual-issue with floating point and an integer vector unit) and 8MBytes of memory. A pair of processors is expected to occupy about 20% of the area of one tile's memory making this chip just slightly larger than a 4Gb DRAM chip built using the same technology[11]. The chip also has a larger pin-count than a straight DRAM chip to accommodate the multiple router channels.

9. Here we mean *physical* locality, rather than the more conventional notions of *spatial* and *temporal* locality that refer to address streams, not to physical locations.
10. Cost here is silicon area divided by yield. This reflects the true recurring cost of manufacturing and ignores the large difference in margin between commodity memories and certain (but not all) processors.

The data in Table 2 illustrate the need to exploit locality in machines of this sort. The data show that at the tile level locality is motivated by latency while at the chip level locality is motivated by limited bandwidth. The slow wires expected in a 2009 process require 1ns (2 cycles) to transmit a bit across a tile using repeaters. Thus, the round-trip latency for a processor to access memory within its own tile is 2 clock cycles[12] while the round-trip latency to access the most distant memory on the same chip is 56 cycles (corner-to-corner). Unlike modern multiprocessors with their all-or-nothing locality, the latency here varies continuously with distance. Latency is reduced by placing data near their point of use, not just at their point of use. Bandwidth is not an issue on chip. The 90,000 wiring tracks per chip edge in this technology are sufficient to allow every processor to fetch 22 words from arbitrary locations on the chip each clock cycle. Even with a vector unit, it is unlikely that a processor would be able to sustain a demand for more than about 8 words per cycle[13].

Table 2: Characteristics of a 2009 Processor/DRAM Chip

Parameter	Value	Units
Signals per chip	400	
Bandwidth per signal	4	Gb/s
External bandwidth per chip	1.6	Tb/s
External bandwidth per tile	25	Gb/s
External bandwidth per tile	**0.2**	Words/clock
Tracks	90,000	
Bandwidth per signal	2	Gb/s
Global on-chip bandwidth	180	Tb/s
Global on-chip bandwidth per tile	2.8	Tb/s
Global on-chip bandwidth per tile	**22**	Words/clock
Access latency (within a tile)	**2**	clocks
Access latency (corner to corner)	**56**	clocks

Bandwidth is a major issue between chips. Available bandwidth falls abruptly (by a factor of 110) at the boundary of the chip. Applications without data locality will be limited by global bandwidth to operate at a small fraction of peak performance. This bandwidth bottleneck is also the reason why the processors must be integrated on the same chip as the memory. The aggregate bandwidth demand of 64 processors, even without vector units, exceeds the bandwidth of

---

11. We assume that each processor is duplicated to keep the yield of these chips comparable to that of DRAM chips with the same capacity while preserving locality. More efficient redundancy schemes are possible.
12. This assumes that the processor is in the center of the tile, not on the edge as drawn in the figure.
13. One can trade some of this excess bandwidth for latency by making one or more of the metal layers thick with wider than minimum design rules. Assuming a constant clock rate, doubling the width, spacing, and thickness of the wires doubles their velocity, halving both latency and bandwidth.

the chip boundary. A large fraction of the memory references from these processors must be captured on-chip.

The architecture sketched here is more economical than a uniprocessor with the same amount (8GBytes) of memory. Its cost (chip area) is only 20% greater than the uniprocessor[14]. At the same time, it has 1024 processors to apply to the problem rather than one. At any speedup greater than 1.2, the fine-grain computer will be able to solve more problems per dollar× second than a uniprocessor[15]. For typical speedups, in the range of 10-100, the cost-performance advantage is significant. The economy here comes from the use of inexpensive processors to more quickly re-use the expensive memory system.

5. Architecture challenges

Considerable work is needed before the vision of a fine-grain machine sketched above becomes a reality. Two areas in particular demand attention: managing locality and reducing overhead. Methods must be developed to manage data locality for irregular problems with significant global interaction. While the vast majority of real programs have irregular, data-dependent, and often time-varying data structures, most work to date on data layout has addressed only regular data structures. A run-time mechanism to migrate data and tasks that simultaneously balances load and minimizes communication is required. Such a mechanism will involve both hardware, to identify communication patterns and to remap addresses, and software, to implement migration and replication policies. As with much of computer architecture, the art is in defining the right interface between these two components.

To simplify the task of extracting parallelism at fine granularity, communication and synchronization overhead must be reduced to a minimum. Low overhead (a few clock cycles) makes it feasible to make every few loop iterations (50-100 clock cycles) a separate task. Reducing the task size greatly increases the amount of available parallelism and hence makes it easier to parallelize programs. It is also important that synchronization be made as specific as possible. Many parallel programs have poor speedup because they are over synchronized, performing a *barrier* synchronization every loop iteration. By synchronizing on individual data elements, rather than on the flow of control, the end of one loop iteration can be overlapped with the beginning of the next iteration, eliminating a sequential bottleneck and greatly increasing performance. Much progress has already been made on the design of low-overhead mechanisms. The J-Machine is able to perform remote task invocations in 50 clock cycles [NWD93]. The M-Machine offers single-cycle communication and synchronization between on-chip tasks [KDM+98]. More work is needed to further reduce overhead while at the same time offering a simple, abstract model to the programmer.

Locality and overhead are fundamental issues that will determine how the next generation of computer systems will be organized. Such fundamental questions cannot be addressed by the benchmark parameter studies that have become very popular in the computer architecture community. While such parameter studies are good at fine-tuning a machine, they are not suitable for the large scale exploration of the design space where programs must be significantly restructured to exploit a new design.

Tuning is needed for fine-grain machines as well. Once the major issues have been resolved, parameter studies can be conducted to determine the appropriate ratio of processors to memory,

---

14. If the 20% cost is considered too high, one can do nearly as well with 16 processor pairs on a chip, each with 64 MBytes of memory, giving a cost increase of only 5%.
15. As an aside, because it is better able to exploit locality, the fine-grain computer is able to outperform the uniprocessor even on completely sequential programs that are limited by memory latency [Burger97].

to evaluate alternative clustering strategies, and the division of memory resources across the memory hierarchy. Before these 'least significant bits' can be addressed, however, we need to get the 'most significant bits' of the architecture resolved.

6. Solving the software problem

Perhaps the most daunting obstacle to the widespread use of fine-grain parallel computers is the issue of software. Almost all software in use today is sequential or has a small number (2-8) of parallel threads. While it is not difficult to write parallel programs, parallelizing existing serial programs is currently beyond our reach. Until there is a software base of useful applications for fine-grain machines, there will be little demand for such machines, their volume will remain low, and hence their price high. The problem here is twofold: First, we must develop programming systems that simplify the task of parallel programming and second, we must chart a path of migration from sequential systems to fine-grain parallel systems.

A promising approach to the migration problem is to use a fine-grain multicomputer as a *smart memory* for a conventional processor. This memory will cost slightly (about 20%) more than a conventional memory system and for most programs will behave exactly like an ordinary memory[16]. The kernels of several key applications can then be ported to the fine-grain machine, one at a time, to realize the performance advantage. Such a machine would offer both compatibility, it runs all of the old software, and performance, by running a few key applications an order of magnitude faster than the conventional machine. Such machines would gain popularity in niche areas (e.g., circuit simulation, logic simulation, and image analysis) as applications are ported. Eventually, we expect that all applications (even sequential applications) would run on the fine-grain machine and the conventional processor would become a vestigial organ. The model here is similar to that used for recently introduced SIMD instruction-set extensions such as MMX. Old programs run without using MMX but run faster when converted to use MMX.

Many parallel programming systems today burden the programmer with incidental issues and bookkeeping tasks. With costly communication and synchronization mechanisms, the programmer is tasked with aggregating computation to increase the grain size and minimize the use of the expensive mechanisms. Systems that use explicit messaging require the programmer to manage the coherence of shared data manually. Without adequate synchronization mechanisms systems must either oversynchronize with costly barriers or run the risk of data races. With coarse mechanisms the programmer has to carefully tune which tasks are to be run in parallel since the overhead associated with a poor choice results in a slower program.

Efficient hardware and software mechanisms free the programmer from such incidental concerns allowing her to concentrate on the essential issues of program decomposition, locality, and load balance. With efficient communication and synchronization, the program can be decomposed at a natural grain size (often quite fine) without concern for overhead. A good memory system automatically manages the replication and coherence of data while allowing the programmer to specify pragmas that minimize the communication required. With such mechanisms the programmer can focus on finding a decomposition that maximizes parallelism while minimizing communication bandwidth and memory required. Without good underlying mechanisms both the programmer and the programming tools (e.g., parallelizing compilers) spend their time dealing with artificial problems of overhead rather than solving the real problems of discovering parallelism, exploiting locality, and balancing load.

---

16. Note that such a memory is by no means pin compatible with standard DRAMs. The physical design of such a memory system is dominated by the inter-chip network of the multicomputer. To the software on the conventional machine, however, the multicomputer appears as memory.

The good news is that even at modest problem sizes (10MByte - 1Gbyte) there is abundant parallelism in almost all demanding problems. Graphics and image processing problems have parallelism that scales as the number of objects (10^4-10^6) and the number of pixels (10^6). Directly solving sparse linear systems of equations has parallelism that scales with the square root of the number of unknowns (10^2-10^3). Evaluating circuit models scales linearly with the number of models (10^3-10^5). Compiled logic simulation has parallelism that scales as the number of gates divided by the number of levels (10^4-10^6). There is sufficient parallelism in all of these problems to keep the 1024 processors in the machine ofSection 4 productively employed. To expose the parallelism in these problems we need hardware and software mechanisms that allow efficient fine-grain communication and synchronization and that automate the bookkeeping tasks associated with coherence and task management. With the appropriate tools, writing parallel programs need not be significantly harder than writing sequential programs.

7. Conclusion

Twenty years ago the case for applying VLSI technology to build parallel machines appeared compelling. Anticipating a 1000-fold increase in number of devices per chip over two decades considerable research focused on ways to realize arrays of simple processors. Instead, however, the increased density of VLSI technology has been largely applied to closing the 100-fold performance gap that existed between microprocessors and high-end CPUs in 1979. Parallel machines did not emerge (except at the high end) because designers were able to realize significant performance gains applying the additional devices to uniprocessors that were able to run existing software.

The situation today is quite different. We again anticipate a 1000-fold increase in the number of devices over the next two decades. This time, however, there is no gap between microprocessors and high-end CPUs (they are one in the same) and we are well beyond the point of diminishing returns in applying devices or grids to increase the performance of single-sequence machines. Applying the increased device count to build explicitly parallel machines appears to be the only alternative. Because a large increase in performance results from a small increment in chip area, such machines are more economical, able to solve more problems per dollar×second, than today's memory-dominated uniprocessors. We are enabled to build such machines because the research of the last 20 years has taught us how to build fast networks, efficient mechanisms, and scalable shared memory.

We envision computing systems of all sizes being built from combined processor-DRAM chips with about 20% of their area devoted to a number of simple, powerful processors. In a 2009 technology such a chip will contain 64 2GHz dual-issue 64-bit processors, each with 8MBytes of memory. Each processor has device count applied to increase its performance to, but not beyond, the point of diminishing returns. At a given memory size such a machine will take 20% more chip area than a conventional computer system but offer orders of magnitude better performance on demanding problems.

While VLSI architecture research over the last 20 years has laid the groundwork for these machines, much work remains. Three areas in particular demand attention: managing locality, reducing overhead, and offering a smooth transition from sequential computing to fine-grain parallel computing. Particularly for irregular applications with dynamic communication patterns hardware and software mechanisms are required to manage the placement and migration of data to minimize use of scarce off-chip bandwidth and avoid long on-chip latencies. To simplify programming, communication and synchronization overhead must be reduced to the point that pro-

grammers can concentrate on the fundamental issues of parallelism, locality, and load balance rather than optimizing the use of expensive mechanisms or performing bookkeeping tasks.

Software has always represented the largest barrier to new architectures. The transition from sequential machines to fine-grain parallel machines can be facilitated by using fine-grain machines as *smart memories* for conventional machines. Such a hybrid machine runs existing sequential applications and provides a platform for parallelizing the critical portions of key applications using the efficient fine-grain mechanisms. Unlike 20 years ago, hardware vendors, software vendors, and customers are motivated to make this transition because there are few alternatives to continue the exponential increase in processor performance with time.

8. Acknowledgments

The authors are indebted to Mark Horowitz and Kunle Olukotun for many helpful discussions. The work of the many students and staff involved in the MIT J-Machine and M-Machine projects greatly influenced the thoughts presented here. This work was supported by DARPA under ARPA order E254 and monitored by the Army under contract DABT63-96-C-0037.

9. References

[ACP95] Anderson, T., Culler, D., and Patterson, D., "A Case for Networks of Workstations: NOW", *IEEE Micro*, Feb 1995, pp. 54-59.

[ASHH88] Agrawal, A., R. Simoni, J. Hennessy, and M. Horowitz, "An Evaluation of Directory Schemes for Cache Coherence," *ISCA-15*, 1988. pp. 280-289.

[Brown79] Browning, S., "Computations on a Tree of Processors", *Caltech Conference on Very-Large-Scale Integration*, 1979, pp. 453-478.

[BKG97] Burger, D., Kaxiras, S., and Goodman, J, "DataScalar Architectures," *ISCA-24*, 1997, pp. 338-349.

[Dally86] Dally, W., "Directions in Concurrent Computing," *ICCD-86*, 1986, pp. 102-106.

[Dally92] Dally, W., "Virtual-Channel Flow Control," *IEEE TPDS*, 3(2), March 1992, pp. 194-205.

[Davis79] Davis, A., "A Data-Driven Machine Architecture Suitable for VLSI Implementation", *Caltech Conference on Very-Large-Scale Integration*, 1979, pp. 479-494.

[DYN97] Duato, J. Yalamanchili, S., and Ni, L., *Interconnection Networks: An Engineering Approach*, IEEE Computer Society Press, 1997.

[FKD+95] Fillo, M., Keckler, S., Dally, W., Carter, N., Chang, A., Gurevich, Y., and Lee, W., "The M-Machine Multicomputer," *Micro-28*, 1995, pp. 146-156.

[KDM+98] Keckler, S., Dally, W., Maskit, D., Carter, N., Chang, A., and Lee, W, "Exploiting Fine-Grain Thread-Level Parallelism on the MIT Multi-ALU Processor", *ISCA-25*, 1998, pp. 306-317.

[LanRus71] Landman, B., and Russo, R., "On a Pin vs. Block Relationship for Partitioning of Logic Graphs," *IEEE TOC*, C-20(12), Dec. 1971, pp. 1469-1479.

[LDK+98] Lee, W., Dally, W., Keckler, S., Carter, N., and Chang, A., "Efficient, Protected Message Interface in the MIT M-Machine," *IEEE Computer*, Nov. 1998, pp. 69-75.

[Mead79] Mead, C., "VLSI and Technological Innovation", *Caltech Conference on Very-Large-Scale Integration*, 1979, pp. 15-29.

[Mago79] Mago, G., "A Cellular, Language-Directed Computer Architecture", *Caltech Conference on Very-Large-Scale Integration*, 1979, pp. 447-452.

[NWD93] Noakes, M.D., Wallach, D.A., and Dally, W.J., "The J-Machine Multicomputer: An Architectural Evaluation," *ISCA-20*, 1993, pp. 224-235.

[Russell78] Russel, R. M., "The Cray-1 Computer System," *Communications of the ACM*, 21(1), January 1978, pp. 63-72.

[SBSS93] Seitz, C., Boden, N. Seizovic, J. and Su W-K., "The Design of the Caltech Mosaic C Multicomputer," *Research in Integrated Systems*, MIT Press, 1993, pp. 1-22.

[Scott96] Scott, S., "Synchronization and Communication in the T3E Multiprocessor," *ASPLOS-VII*, pp. 26-36.

[Sequin79] Sequin, C., "Single-Chip Computers, The New VLSI Building Blocks", *Caltech Conference on Very-Large-Scale Integration*, 1979, pp. 435-446.

[Seitz85] Seitz, C., "The Cosmic Cube," *CACM*, 28(1), Jan. 1985, pp. 22-33.